Figure 1

Modified SSA-conversion process

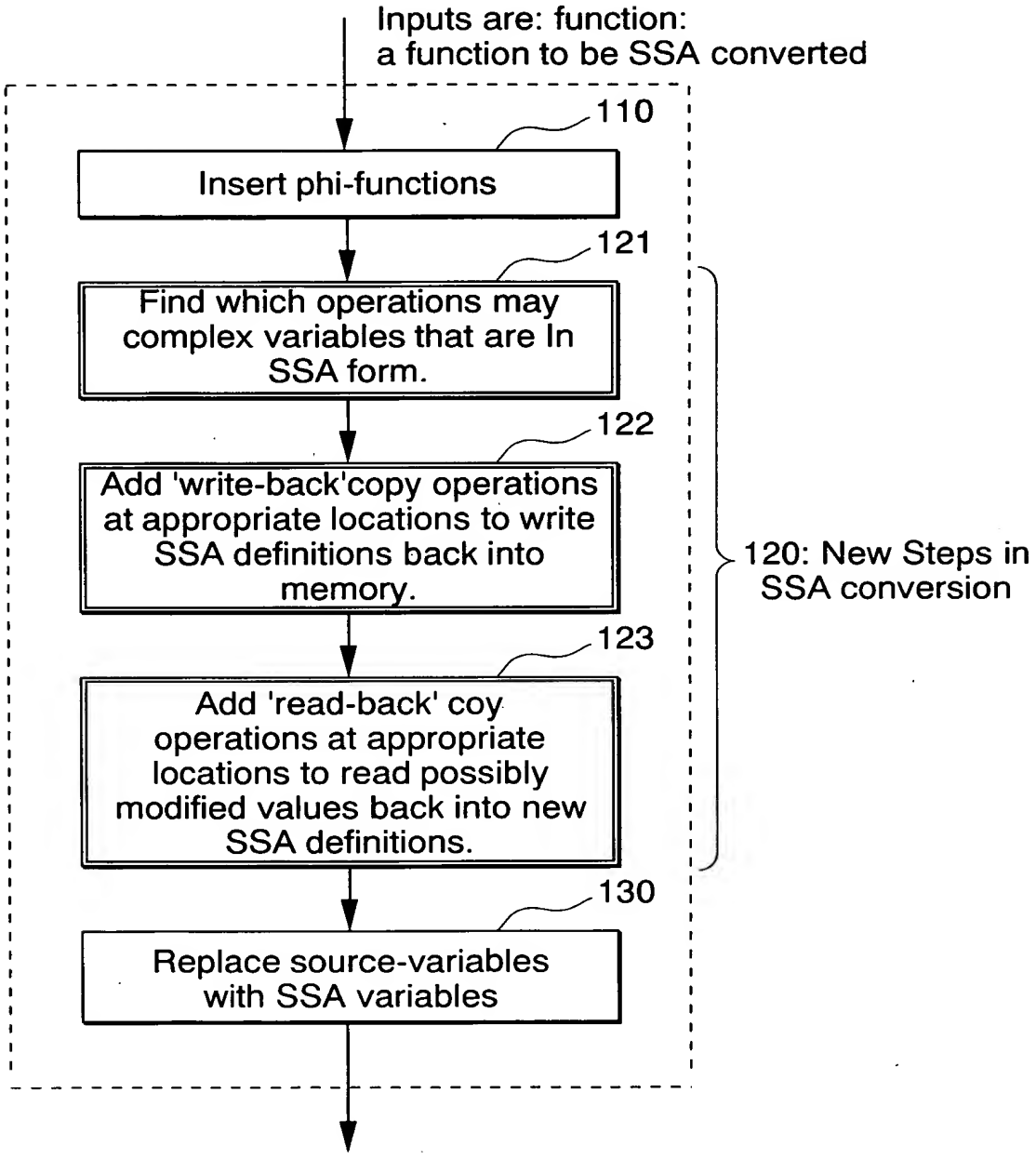

Figure 2

Overall compiler control flow

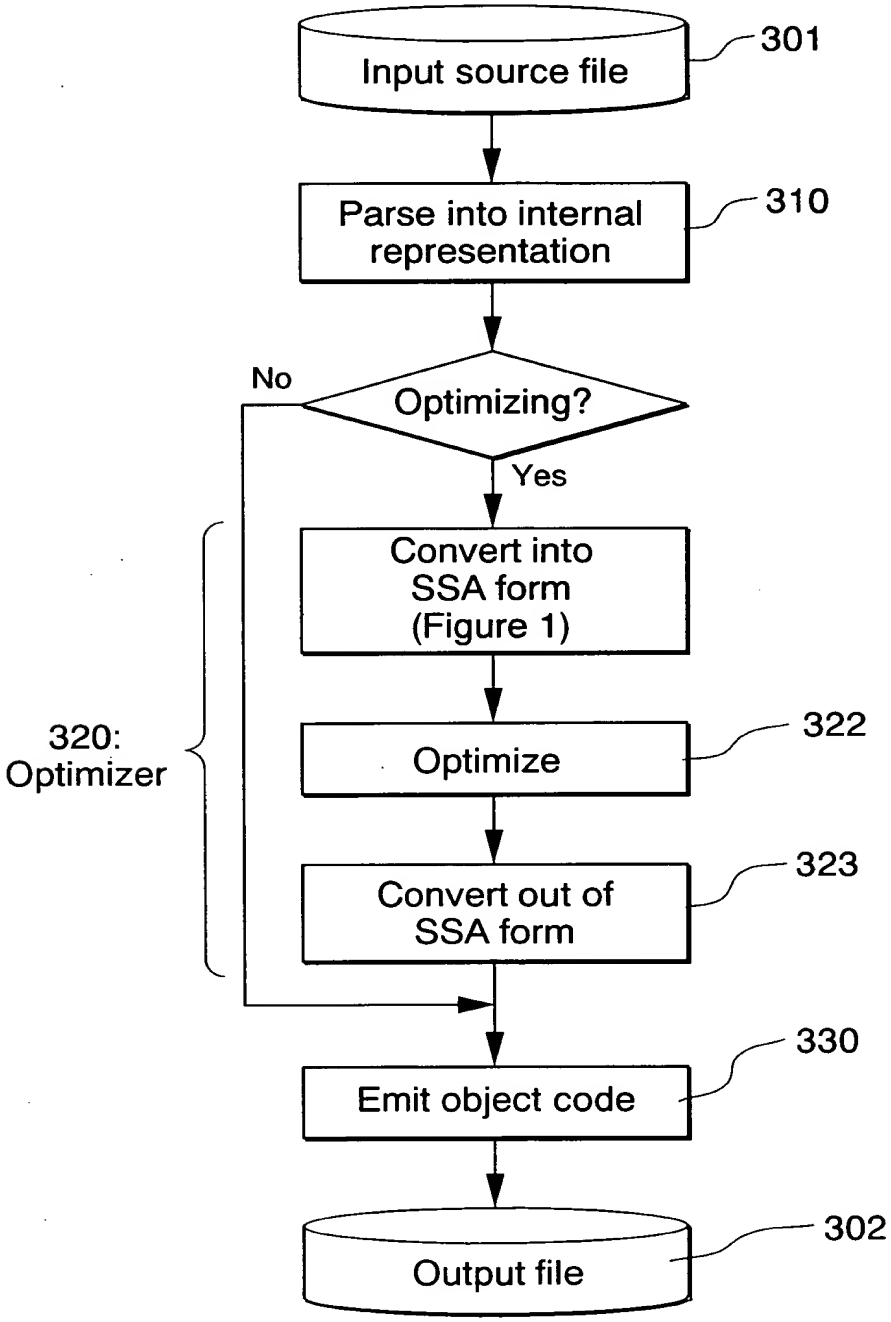

Figure 3
Program representation

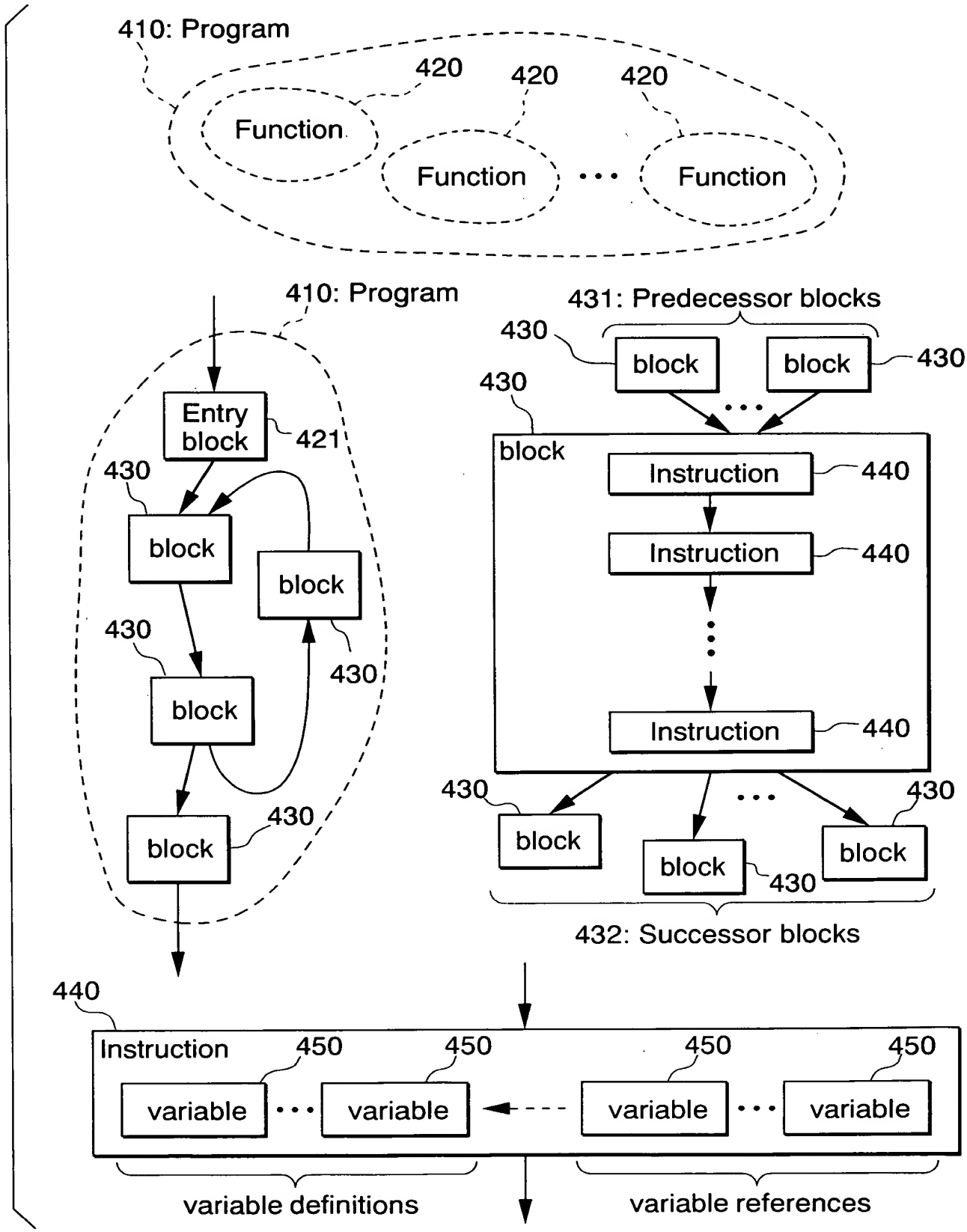

Figure 4

Placement of read/write-backs for the SSA form of *x, (*x)'1

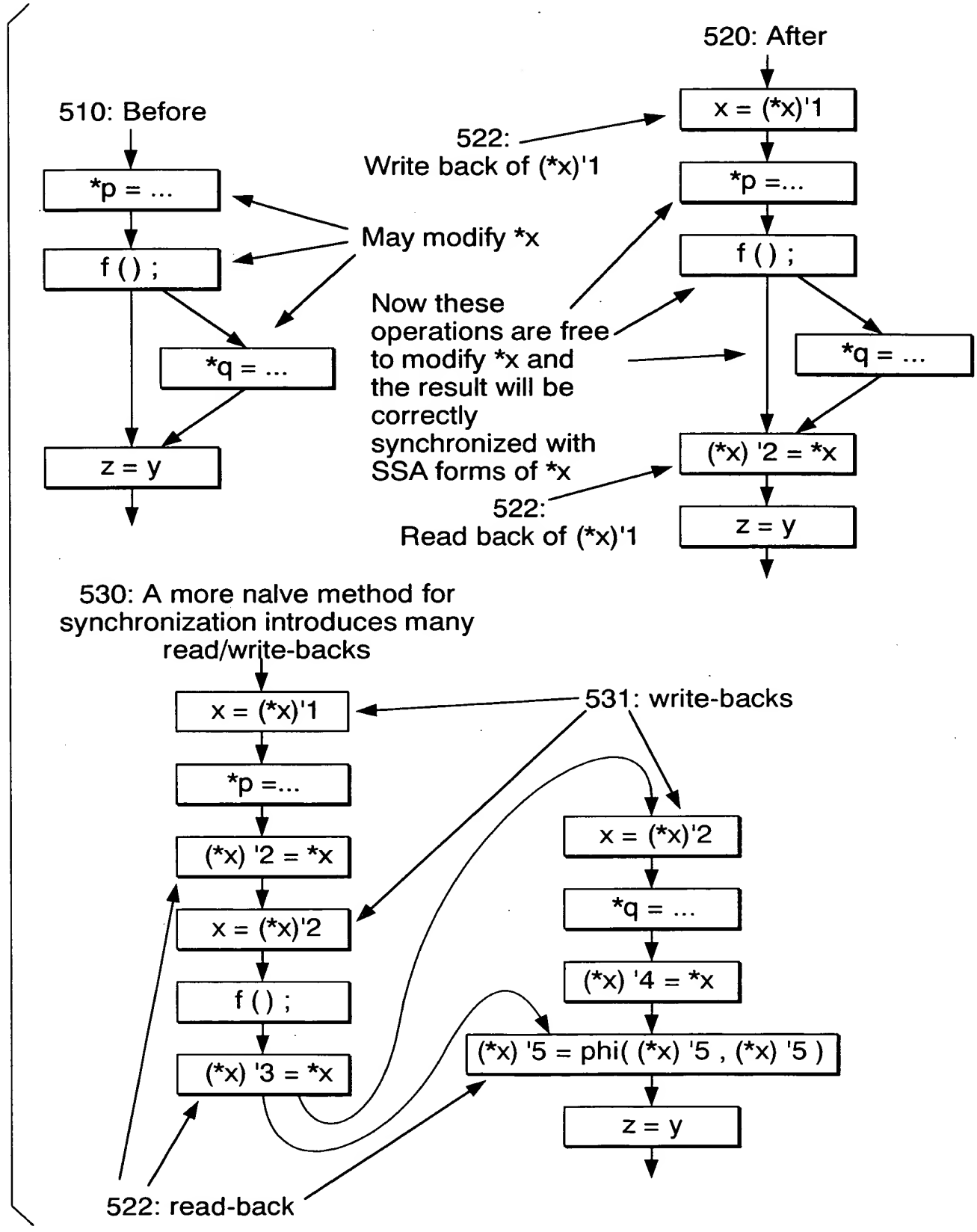

Figure 5

The procedure 'add_syncs_and_write_backs'

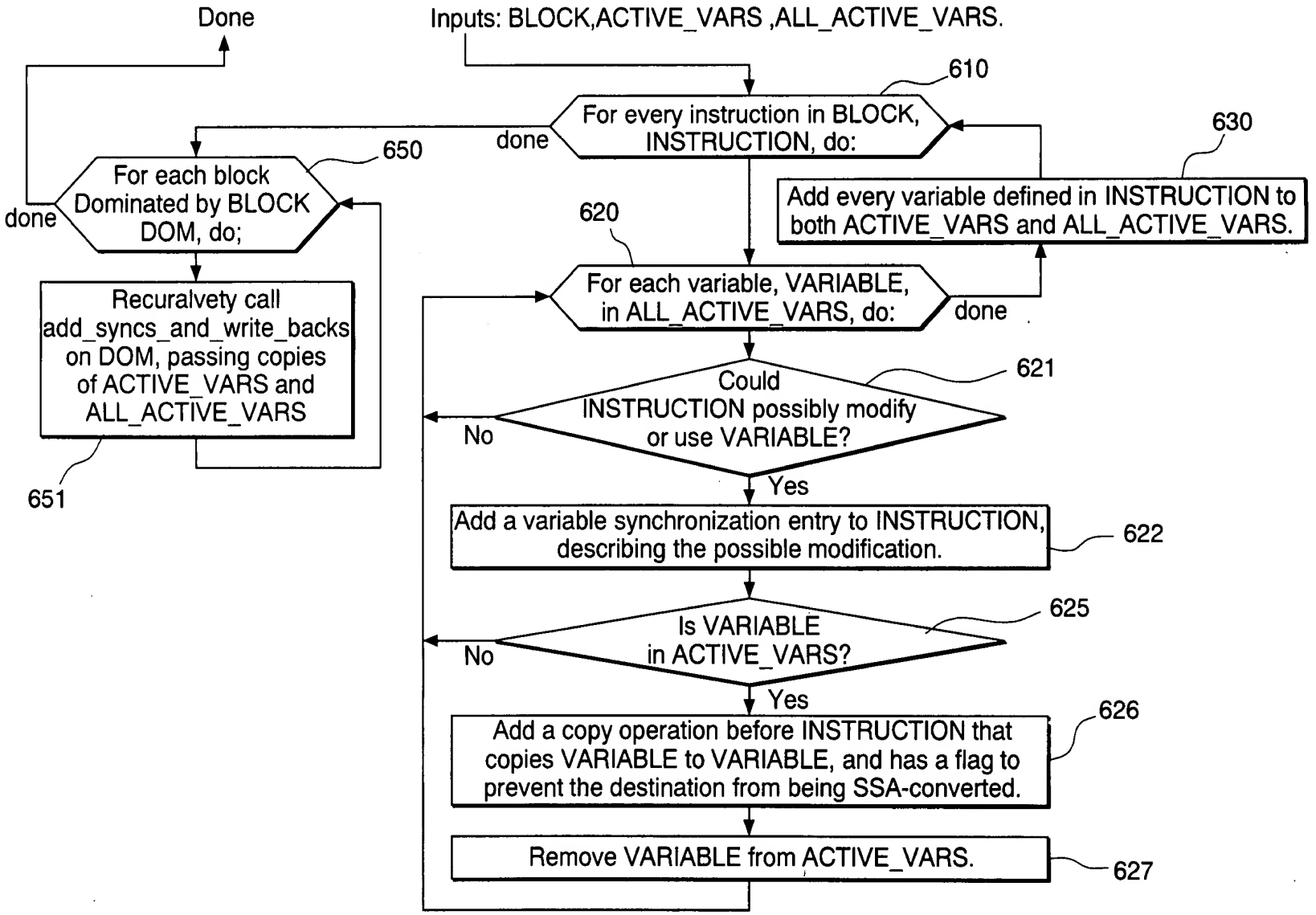

Figure 6

Conversion step (a'.III), insertion of read-backs

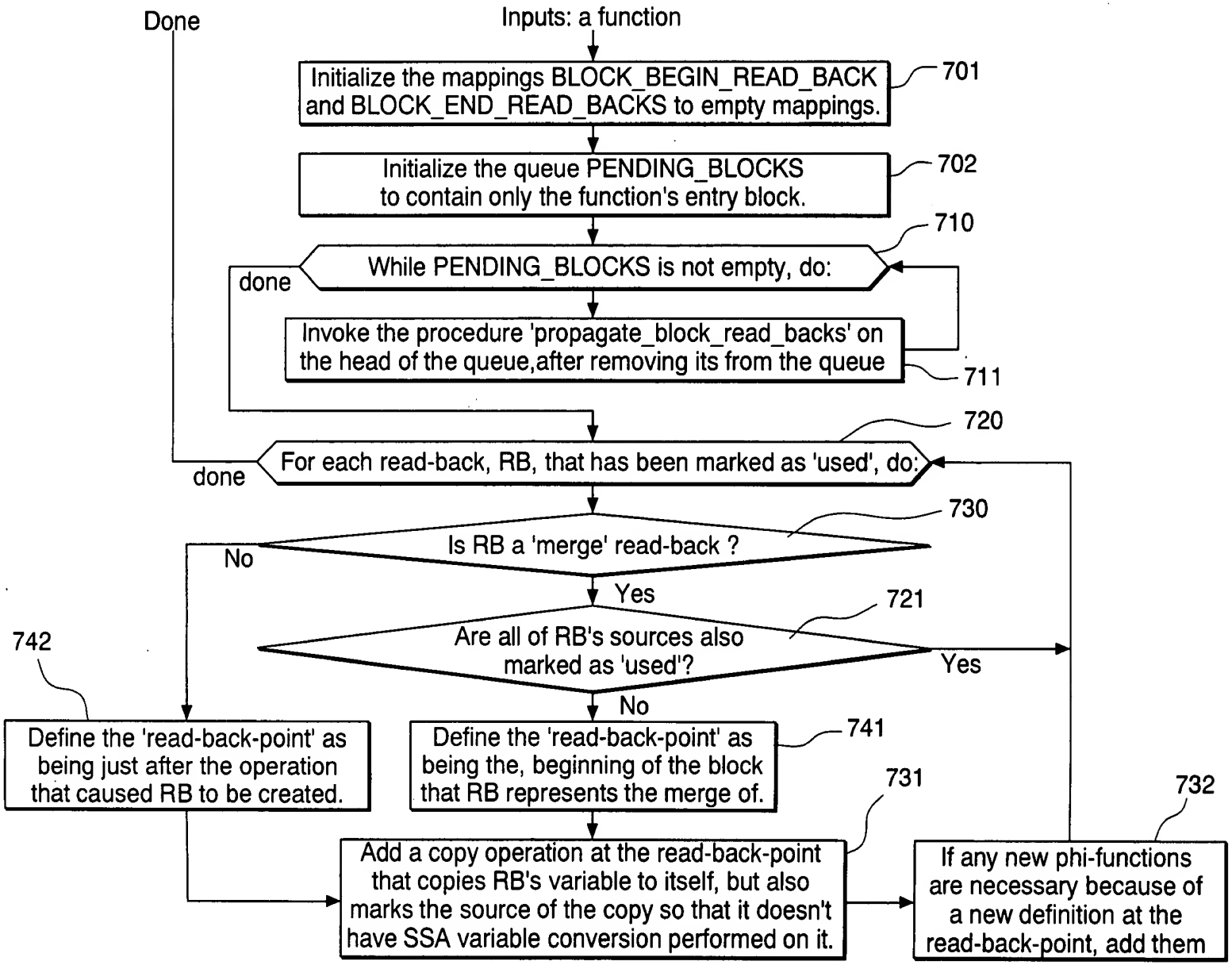

Figure 7A

The procedure 'propagate_read_backs'

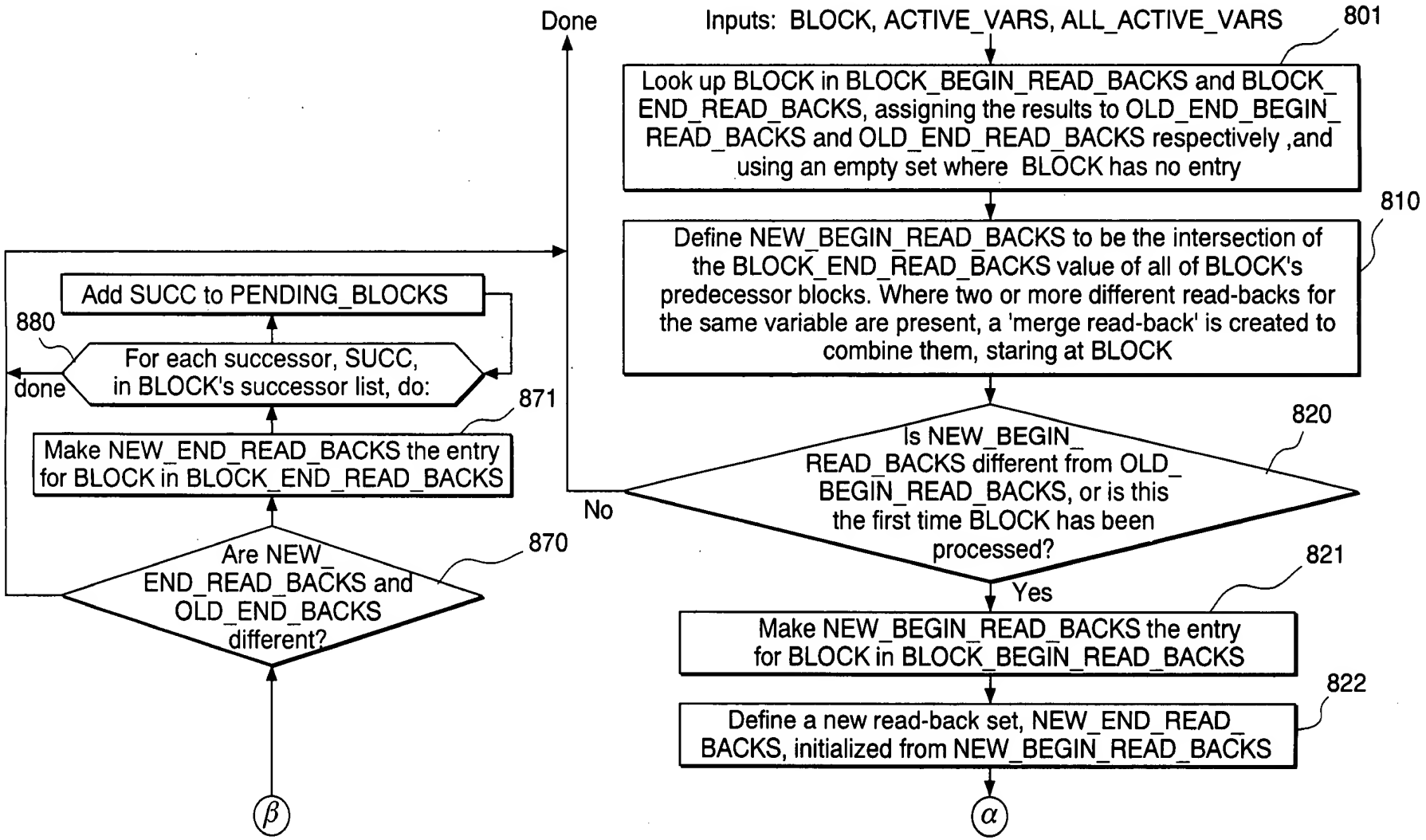

Figure 7B

The procedure 'propagate_read_backs'

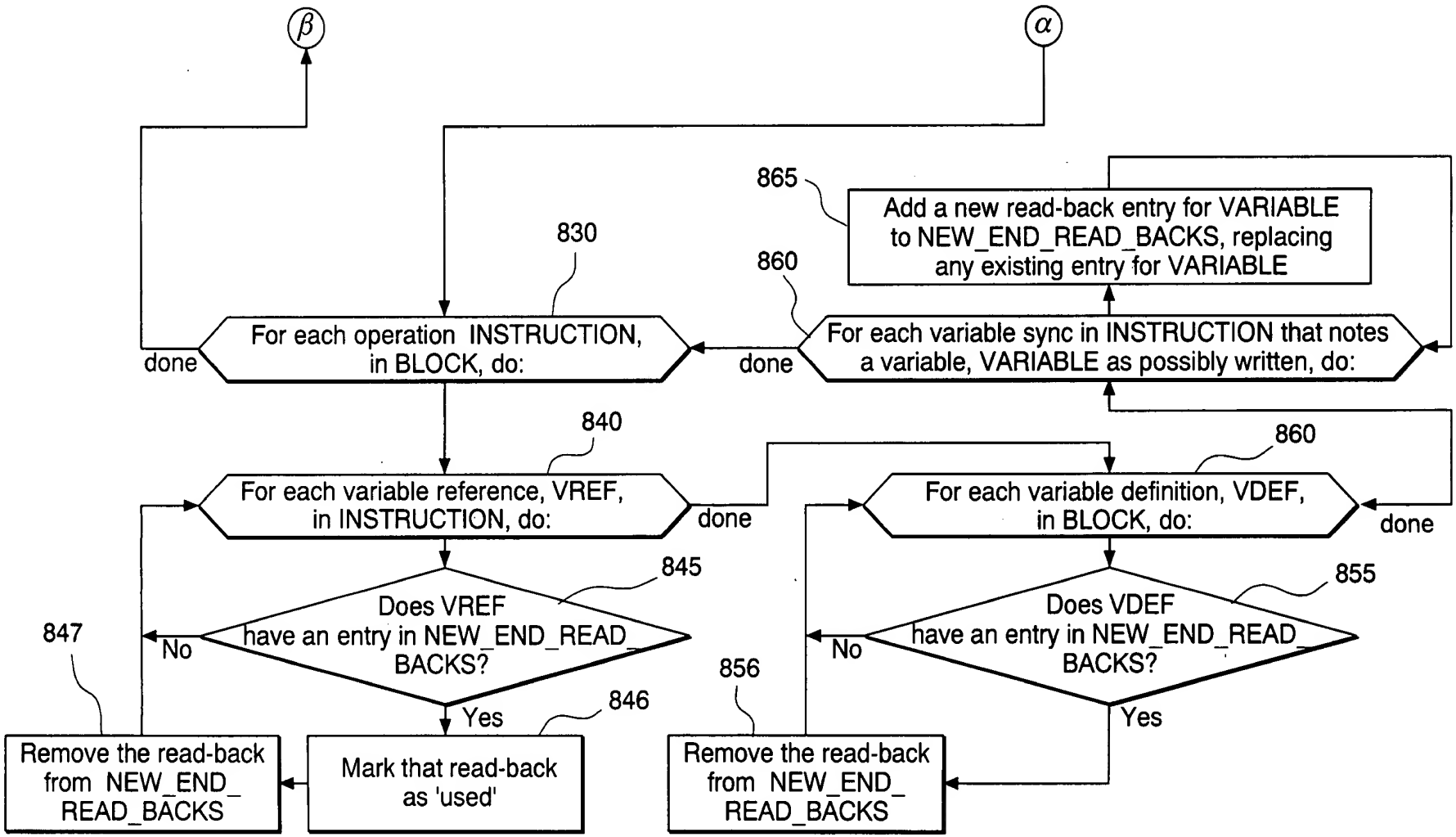

Figure 8
Example source program

This short C program is used to illustrate the invention:

```
extern int g () , h ( ) , i ( ) , x;
int foo (int *p)
{
   (*p) ++;                                                    [810]
   if (*P > 10)
     {
        g ( ) ;
        h () ;
        if (x > 5 )
           g () ;
        if (x > 3)
           i ( );
        else
           X = *p;
        *P = 5;
     }
   return *p;
}
```

Here's the same program converted to a slightly more primitive form:

```
int foo (int *p)
{
block1:
   *p := *p +1;                                                [B20]
   if (*P <= 10)
      goto block8;
block2:
   g () ;
   h () ;
   if (x <= 5)
      goto block4;
block3:
   g () ;
block4:
if (x > 3)
   goto block6;
block5 :
   x :=  *p;                                                   [B40]
   goto block7;
block6 :
   i ();
block7 :
   *p := 5;                                                    [830]
block8:
   return *p;
}
```

Figure 9

SSA converted program, with simple implementation of read-backs :

The following is psuedo-C, augmented with the ` phi' operation, where

RESULT = phi (block1: VAL1 , ..., blockN:VALN)

means ` assign VAL1 to RESULT if control-flow comes from block1 ' , and similarly so on for each value of N.

The extra variables 'pvN", where N is an integer, are SSA versions of *P, and are in fact local variables, not dereferences of p.

```
1nt foo (int *p)
{
  int pvl, pv2 , pv3 , pv4, pv5, pv6;

block1;
  pvl = *P + 1;
  if (pvl <= 10)
    goto block8;
block2:
  *P = pvl;              /* This writes-back PV1 to *P. */
  g ();
  pv2 = *P;              /* This reads-back *P into PV2. */
  *P = pv2;              /* This writes-back PV2 to *P. */
  h ( ) ;
  pv3 = *P;              /* This reads-back *P into PV3 */                    [912]
  if (x <= 5)
    goto block4;
block3:
  *p = pv3;              /* This writes-back PV4 to *p, */
  g ( ) ;
  pv4= *p;               /* This reads-back *p into PV4. */                   [911]
block4:
  pv5 = phi (block3: pv4, block2: pv3)                                        [910]
  if (x > 3)
    goto block6;
block5:
  gota block7;
block6;
  i ( ) ;
block7:
  x = phi (block6: x, block5: pv5);
block8:
  pv6 = phi (block1: pv1,  block7: 5);
  *P = pv6;              /* This writes-back PV6 to *P. */
  return pv6;
}
```

Figure 10

SSA converted program, with the implementation of read-backs described in this patent

```
int foo (int *p)
{
   int pv1, pv2, pv3;

block1:
   pv1 = *p +1;                                                    [1011]
   if (pvl <= 10)
      goto block8;

block2;
   *p = pv1;            /* This writes-back pv1 to *P. */
   g ( ) ;                                                         [1021]
   h ( ) ;                                                         [1022]
   if (x <= 5)
      goto block4;

block3;
   g ();                                                           [1023]

block4:
   pv2 = *p;            /* This reads-back *p into pv2, */         [1030]
   if (x > 3)
      goto block6;

block5;
   goto block7;

block6:
   i ( );                                                          [1024]

black7 :
   x = phi (block6 : x, block5 : pv2) ;                            [1031]

block8:
   pv3 = phi (block1: pv1, block7: 5);                             [1010]
   *P = pv3;            /*This writes-back PV3 to *P */

   return pv3;
}
```

Figure 11

Register-alloced and SSA-unconverted program

using BBA-form requires having a good register allocator that will merge variables where possible, as it tends to generate a lot of variables with short lifetimes. We assume that here.

```
int foo (int *p)
{
   int pv;

block1;
   pv = *p + 1;
   if (pv <= 10)
      goto block8;

block2:
   'P = pv;            /* This writes-back pv to *P. */
   g ( ) ;
   h ( ) ;
   if (x <= 5)
      goto block4;

block3 :
   g ( ) ;

block4 :
   if (x > 3)
      goto block6;

block5:
   x=*p;
      goto block7;

block6:
   i ();

block7:
    pv =5;

block8:
   *P= pv            /* This writes-back PV to *P. */

return pv;
}
```

Figure 12

Original SSA-conversion process